CMSC 330: Organization of Programming Languages

Logic Programming with Prolog
Background

- 1972, University of Aix-Marseille

- Original goal: Natural language processing

- At first, just an interpreter written in Algol
  - Compiler created at Univ. of Edinburgh
More Information On Prolog

- Various tutorials available online
- Links on webpage
Logic Programming

- At a high level, logic programs model the relationship between objects
  1. Programmer specifies relationships at a high level
  2. Programmer specifies basic facts
     - The facts and relationships define a kind of database
  3. Programmer then queries this database
  4. Language searches the database for answers
Features of Prolog

- **Declarative**
  - Facts are specified as *tuples*, relationships as *rules*
  - Queries stated as goals you want to prove, not (necessarily) how to prove them

- **Dynamically typed**

- **Several built-in datatypes**
  - Lists, numbers, records, … but no functions

Prolog not the only logic programming language

- Datalog is simpler; CLP and λProlog more feature-ful
- Erlang borrows some features from Prolog
A Small Prolog Program – Things to Notice

Use /* */ for comments, or % for 1-liners

Periods end statements

Lowercase denotes atoms

Program statements are facts and rules

Uppercase denotes variables

/* A small Prolog program */

% facts:
female(alice).
male(bob).
male(charlie).
father(bob, charlie).
mother(alice, charlie).

% rules for “X is a son of Y”
son(X, Y) :- father(Y, X), male(X).
son(X, Y) :- mother(Y, X), male(X).
Running Prolog (Interactive Mode)

Navigating location and loading program at top level

?- working_directory(C,C).
C = 'c:/windows/system32/'.

?- working_directory(C,'c:/Users/me/desktop/p6').
C = 'c:/Users/me/desktop/'.

?- ['01-basics.pl'].
% 01-basics.pl compiled 0.00 sec, 17 clauses
true.

?- make.
true.
Running Prolog (Interactive Mode)

Listing rules and entering queries at top level

?- listing(son).
son(X, Y) :-
    father(Y, X),
    male(X).
son(X, Y) :-
    mother(Y, X),
    male(X).
true.

?- son(X,Y).
X = charlie,
Y = bob;
X = charlie,
Y = alice.

List rules for son

User types ; to request additional answer

Multiple answers

User types return to complete request
Quiz #1: What is the result?

Facts:
- hobbit(frodo).
- hobbit(samwise).
- human(aragorn).
- human(gandalf).

Query:
?- human(Z).

A. Z=aragorn
B. Z=aragorn; Z=gandalf.
C. Z=gandalf.
D. false.
Quiz #1: What is the result?

Facts:

- hobbit(frodo).
- hobbit(samwise).
- human(aragorn).
- human(gandalf).

Query:

?- human(Z).

A. Z=aragorn
B. Z=aragorn; Z=gandalf.
C. Z=gandalf.
D. false.
Quiz #2: What are the values of Z?

Facts:

hobbit(frodo).
hobbit(samwise).
human(aragorn).
human(gandalf).
taller(gandalf, aragorn).
taller(X,Y) :-
    human(X), hobbit(Y).

Query:

?- taller(gandalf,Z).

A. aragorn
B. frodo; samwise.
C. gandalf; aragorn.
D. aragorn; frodo; samwise.
Quiz #2: What are the values of Z?

Facts:
- hobbit(frodo).
- hobbit(samwise).
- human(aragorn).
- human(gandalf).
- taller(gandalf, aragorn).
- taller(X,Y) :- human(X), hobbit(Y).

Query:
- ?- taller(gandalf,Z).

A. aragorn
B. frodo; samwise.
C. gandalf; aragorn.
D. aragorn; frodo; samwise.
Outline

- Syntax, terms, examples
- Unification
- Arithmetic / evaluation
- Programming conventions
- Goal evaluation
  - Search tree, clause tree
- Lists
- Built-in operators
- Cut, negation
Prolog Syntax and Terminology

Terms

• Atoms: begin with a lowercase letter
  horse  underscores_ok  numbers2
• Numbers
  123  -234  -12e-4
• Variables: begin with uppercase or _  "don’t care" variables
  X  Biggest_Animal  _the_biggest1
  _
• Compound terms: functor(arguments)
  bigger(horse, duck)
  bigger(X, duck)
  f(a, g(X, _), Y, _)

No blank spaces between functor and (arguments)
Clauses (aka statements)

- Facts: define predicates, terminated by a period
  bigger(horse, duck).
  bigger(duck, gnat).
Intuitively: “this particular relationship is true”

- Rules: head :- body
  is_bigger(X,Y) :- bigger(X,Y).
  is_bigger(X,Y) :- bigger(X,Z), is_bigger(Z,Y).
Intuitively: “Head if Body”, or “Head is true if each of
the subgoals in the body can be shown to be true”

A program is a sequence of clauses
Program Style

One predicate per line

blond(X) :-
    father(Father, X),
    blond(Father), % father is blond
    mother(Mother, X),
    blond(Mother). % and mother is blond

Descriptive variable names

Inline comments with % can be useful
Prolog Syntax and Terminology (cont.)

Queries

• To “run a program” is to submit queries to the interpreter
• Same structure as the body of a rule
  ➢ Predicates separated by commas, ended with a period
• Prolog tries to determine whether or not the predicates are true

?- is_bigger(horse, duck).
?- is_bigger(horse, X).

“Does there exist a substitution for X such that is_bigger(horse,X)?”
Two terms unify if and only if

- They are identical
  
  ```prolog
  ?- gnat = gnat.
  true.
  ```

- They can be made identical by substituting variables
  
  ```prolog
  ?- is_bigger(X, gnat) = is_bigger(horse, gnat).
  X = horse.  \text{This is the substitution: what X must be for the two terms to be identical.}
  ```

  ```prolog
  ?- pred(X, 2, 2) = pred(1, Y, X)
  false.
  ```

  ```prolog
  ?- pred(X, 2, 2) = pred(1, Y, _)
  X = 1,
  Y = 2.
  ```

Sometimes there are multiple possible substitutions; Prolog can be asked to enumerate them all.
The = Operator

- For unification (matching)
  
  `?- 9 = 9.`
  
  `true.`
  
  `?- 7 + 2 = 9.`
  
  `false.`

- Why? Because these terms do not match
  
  - 7+2 is a compound term (e.g., +(7,2))

- Prolog does not evaluate either side of =
  
  - Before trying to match
The is Operator

- For arithmetic operations
- LHS is RHS
  - First evaluate the RHS (and RHS only!) to value V
  - Then match: LHS = V
- Examples
  
  ?- 9 is 7+2.  
  true.  
  
  ?- 7+2 is 9.  
  false.  
  
  ?- X = 7+2.  
  X = 7+2.  
  
  ?- X is 7+2.  
  X = 9.
The == Operator

- For identity comparisons
- \( X == Y \)
  - Returns true if and only if \( X \) and \( Y \) are identical
- Examples
  
<table>
<thead>
<tr>
<th>- Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>?- 9 == 9.</td>
<td>true.</td>
</tr>
<tr>
<td>?- X == 9.</td>
<td>false.</td>
</tr>
<tr>
<td>?- X == X.</td>
<td>true.</td>
</tr>
<tr>
<td>?- 9 == 7+2.</td>
<td>false.</td>
</tr>
<tr>
<td>?- X == Y.</td>
<td>false.</td>
</tr>
<tr>
<td>?- 7+2 == 7+2.</td>
<td>true.</td>
</tr>
</tbody>
</table>
The \texttt{=:=} Operator

- For arithmetic operations
- “LHS \texttt{=:=} RHS”
  - Evaluate the LHS to value $V_1$ (Error if not possible)
  - Evaluate the RHS to value $V_2$ (Error if not possible)
  - Then match: $V_1 = V_2$

Examples

\begin{itemize}
  \item \texttt{- 7+2 =:= 9.} \hspace{1cm} \texttt{- 7+2 =:= 3+6.}
    \hspace{1cm} \texttt{true.} \hspace{1cm} \texttt{true.}
  \item \texttt{- X =:= 9.} \hspace{1cm} \texttt{- X =:= 7+2}
    \hspace{1cm} \texttt{Error: =:=/2: Arguments are not sufficiently instantiated}
\end{itemize}
Quiz #3: What does this evaluate to?

Query:

?- 9 = 7+2.

A. true
B. false
Quiz #3: What does this evaluate to?

Query:

?- 9 = 7+2.

A. true
B. false
No Mutable Variables

- = and is operators do not perform assignment
  - Variables take on exactly one value ("unified")

Example
- foo(...,X) :- ... X = 1,... % true only if X = 1
- foo(...,X) :- ... X = 1, ..., X = 2, ... % always fails
- foo(...,X) :- ... X is 1,... % true only if X = 1
- foo(...,X) :- ... X is 1, ..., X is 2, ... % always fails

X can’t be unified with 1 & 2 at the same time
Function Parameter & Return Value

- Code example

increment(X,Y) :-
    Y is X+1.
?- increment(1,Z).
Z = 2.
?- increment(1,2).
true.
?- increment(Z,2).
ERROR: incr/2: Arguments are not sufficiently instantiated to int

Can’t evaluate X+1 since X is not yet instantiated to int
Function Parameter & Return Value

- Code example

addN(X,N,Y) :-
  Y is X+N.

?- addN(1,2,Z).
Z = 3.
Recursion

- **Code example**
  
  \[\text{addN}(X,0,X).\]
  
  \[\text{addN}(X,N,Y) : - X1 \text{ is } X+1,\]
  \[\text{N1} \text{ is } N-1,\]
  \[\text{addN}(X1,N1,Y).\]
  
  \[?- \text{addN}(1,2,Z).\]
  \[Z = 3.\]

  *Base case*

  *Inductive step*

  *Recursive call*
Quiz #4: What are the values of X?

Facts:

mystery(_,0,1).
mystery(X,1,X).
mystery(X,N,Y) :-
  N > 1,
  X1 is X*X,
  N1 is N-1,
mystery(X1,N1,Y).

Query:

?- mystery(5,2,X).

A. 1.
B. 32.
C. 25.
D. 1; 25.
Quiz #4: What are the values of X?

Facts:
mystery(_,0,1).
mystery(X,1,X).
mystery(X,N,Y) :-
    N > 1,
    X1 is X*X,
    N1 is N-1,
    mystery(X1,N1,Y).

Query:
?- mystery(5,2,X).

A. 1.
B. 32.
C. 25.
D. 1; 25.
Factorial

Code

factorial(0,1).
factorial(N,F) :-
    N > 0,
    N1 is N-1,
    factorial(N1,F1),
    F is N*F1.
Tail Recursive Factorial w/ Accumulator

- **Code**

  ```prolog
tail_factorial(0,F,F).
tail_factorial(N,A,F) :-
    N > 0,
    A1 is N*A,
    N1 is N - 1,
    tail_factorial(N1,A1,F).
```
And and Or

- **And**
  - To implement $X \land Y$ use `,` in body of clause
  - E.g., for $Z$ to be true when $X$ and $Y$ are true, write $Z : - X,Y$.

- **Or**
  - To implement $X \lor Y$ use two clauses
  - E.g., for $Z$ to be true when $X$ or $Y$ is true, write $Z : - X$. $Z : - Y$. 
Goal Execution

When submitting a query, we ask Prolog to substitute variables as necessary to make it true.

- Prolog performs **goal execution** to find a solution
  - Start with the goal, and go through statements in order
  - Try to unify the head of a statement with the goal
  - If statement is a rule, its hypotheses become subgoals
    - Substitutions from one subgoal constrain solutions to the next
  - If goal execution reaches a dead end, it **backtracks**
    - Tries the next statement
  - When no statements left to try, it reports **false**

- More advanced topics later – cuts, negation, etc.
Consider the following:

- “All men are mortal”
  \[ \text{mortal}(X) :\sim \text{man}(X). \]
- “Socrates is a man”
  \[ \text{man}(\text{socrates}). \]
- “Is Socrates mortal?”
  \[ ?- \text{mortal}(\text{socrates}). \]
  \[ \text{true}. \]

How did Prolog infer this?

1. Sets \( \text{mortal}(\text{socrates}) \) as the initial goal.
2. Sees if it unifies with the head of any clause:
   \[ \text{mortal}(\text{socrates}) = \text{mortal}(X). \]
3. \( \text{man}(\text{socrates}) \) becomes the new goal (since \( X = \text{socrates} \)).
4. Recursively scans through all clauses, backtracking if needed …
Clause Tree

- Clause tree
  - Shows (recursive) evaluation of all clauses
  - Shows value (instance) of variable for each clause
  - Clause tree is true if all leaves are true

- Factorial example

  factorial(0,1).
  factorial(N,F) :-
      N > 0,
      N1 is N-1,
      factorial(N1,F1),
      F is N*F1.
Clause Tree

- Clause tree
  - Shows (recursive) evaluation of all clauses
  - Shows value (instance) of variable for each clause
  - Clause tree is true if all leaves are true

- Factorial example

factorial(0,1).
factorial(N,F) :-
  N > 0,
  N1 is N-1,
  factorial(N1,F1),
  F is N*F1.

```
factorial(3,6)
  3>0  2 is 3-1
  factorial(2,2)  6 is 3*2
    2>0  1 is 2-1
    factorial(1,1)  2 is 2*1
      1>0  0 is 1-1
      factorial(0,1)  1 is 1*1
        true
```
Tracing

- trace lets you step through a goal’s execution
  - notrace turns it off

```
my_last(X, [X]).
my_last(X, [\_\_T]) :-
  my_last(X, T).

?- trace.
true.

[trace]  ?- my_last(X, [1,2,3]).
  Call: (6) my_last(_G2148, [1, 2, 3]) ? creep
  Call: (7) my_last(_G2148, [2, 3]) ? creep
  Call: (8) my_last(_G2148, [3]) ? creep
  Exit: (8) my_last(3, [3]) ? creep
  Exit: (7) my_last(3, [2, 3]) ? creep
  Exit: (6) my_last(3, [1, 2, 3]) ? creep
X = 3
```
Goal Execution – Backtracking

- Clauses are tried in order
  - If clause fails, try next clause, if available

- Example

  jedi(luke).
  jedi(yoda).
  sith(vader).
  sith( Maul ).
  fight(X,Y) :- jedi(X), sith(Y).

  

  ?- fight(A,B).
  A=luke,
  B=vader;
  A=luke,
  B=maul;
  A=yoda,
  B=vader;
  A=yoda,
  B=maul.
Prolog (Search / Proof / Execution) Tree

?- fight(A,B).

A=X, B=Y

?- jedi(X), sith(Y).

X=luke

?- jedi(luke), sith(Y).

Y=vader

?- sith(vader).

Y=maul

?- sith(maul).

X=yoda

?- jedi(yoda), sith(Y).

Y=vader

?- sith(vader).

Y=maul

?- sith(maul).